

The Application of Hardware in the Loop Testing for Distributed Engine Control

George L. Thomas¹

N&R Engineering and Management Services, Inc., Parma Heights, OH, 44130, USA

Dennis E. Culley²

NASA Glenn Research Center, Cleveland, OH, 44135, USA

and

Alex Brand³

Sporian Microsystems, Inc., Lafayette, CO, 80026, USA

The essence of a distributed control system is the modular partitioning of control function across a hardware implementation. This type of control architecture requires embedding electronics in a multitude of control element nodes for the execution of those functions, and their integration as a unified system. As the field of distributed aeropropulsion control moves toward reality, questions about building and validating these systems remain. This paper focuses on the development of hardware-in-the-loop (HIL) test techniques for distributed aero engine control, and the application of HIL testing as it pertains to potential advanced engine control applications that may now be possible due to the intelligent capability embedded in the nodes.

Nomenclature

BPPD	=	Blade passing pressure disturbance
C-MAPSS40k	=	Commercial modular aeropropulsion system simulation, 40,000 lbf thrust class
DEC	=	Distributed engine control
DECSS	=	Distributed engine control system simulator
FADEC	=	Full authority digital engine control
F_n	=	Net thrust produced by engine
HPC	=	High pressure compressor
HIL	=	Hardware-in-the-loop
HPT	=	High pressure turbine
LPC	=	Low pressure compressor
LPT	=	Low pressure turbine
N_f	=	Low pressure shaft speed (rpm)
N_c	=	High pressure shaft speed (rpm)
P_3	=	Total pressure at station 3 (at HPC exit) (psi)
P_{s3}	=	Static pressure at station 3 (at HPC exit) (psi)
P_{3HB}	=	High bandwidth total pressure at station 3 (psi)
PLA	=	Power lever angle
W_f	=	Fuel mass flow rate (lbm/s)

¹ Controls Engineer, george.l.thomas@nasa.gov

² Research Engineer, dennis.e.culley@nasa.gov, AIAA Senior Member

³ Electrical Engineer, abrand@sporian.com

I. Introduction

THERE is a desire in the jet propulsion control industry to design engines that feature distributed control, which will be implemented using networks of smart transducers. Distributed control has been an exceedingly difficult problem due to the extreme temperature environment of the engine and the subsequent implications for system weight. The barrier has been high-temperature electronics, which has the ability to make the weight trade more favorable. As this high-temperature capability moves toward reality, questions about building and validating these systems remain. In modern production engines, wiring harnesses are typically used to connect the analog signals from control elements (i.e., sensors and actuators) on the engine to the full-authority digital electronic controllers (FADECs) that regulate them. In the case of pressure signals, pneumatic tubes are plumbed from the engine sensing location to the pressure transducer housed in the engine control unit (ECU). These cables and tubes can be long and heavy, and the interconnects can pose reliability concerns, which motivates placing the electronics and transducers as close as possible to their respective stations on the engine.¹ The lack of availability of high-temperature electronics makes it difficult to place these devices in the most desirable locations, however solutions in this field are progressing.¹ This rearrangement of electronics from the FADEC enables digitization at locations closer to their respective signal sources, improving the signal quality and bandwidth. Signal lag is especially characteristic of long runs of pneumatic tubing, and so the advent of distributed engine control will significantly affect the bandwidth of pressure signals that can be measured from the engine. These embedded electronics may thus provide new information for control, thus spurring development of advanced control technologies. Applications such as active surge control and active combustion control may become realizable, because information about the stability of the operating state of various engine components is available in the high frequency content of engine pressure data.

Some applications of the high bandwidth sensing and actuation techniques that will be enabled by distributed engine control (DEC) have already been explored. Bright et al.² used stator vanes with flow injection capabilities to affect the flow separation occurring on these vanes—in that work, the authors observed the correlation of flow separation and the fundamental component of blade passing pressure disturbances, as measured by piezoelectric pressure transducers mounted at the tips of the flow control vanes. Mitigation of stator vane flow separation is useful for reducing loss (increasing efficiency) and postponing the onset of surge or stall. Further, Wadia et al.³ presented an autocorrelation-based metric that captures stall margin information; that work is based on the observation that as a compressor approaches stall, the magnitudes of the high frequency components of the pressure in the compressor will vary in an increasingly erratic fashion. The work published in the current paper seeks to lay a foundation for future tests of various DEC concepts and devices, and so the following subsection will introduce the specifics of this work.

A. Hardware-in-the-Loop Testing of DEC Devices

Relative to traditional hardware-in-the-loop (HIL) simulations, two significant issues must be overcome. First, the number of active control elements has significantly increased in the distributed architecture. In a centralized aeropropulsion control system, the FADEC is the only piece of control logic hardware being tested, as all the analog sensor signals and actuator loads are emulated in the HIL test bed. Distributed control presents a potential multitude of active hardware elements that may or may not be available at any given time. Second, the embedded processing in a smart node provides the potential for high bandwidth capabilities that far exceed the bandwidth of models used for traditional aeroengine control design. Developing methodologies to emulate these wide bandwidth signals in order to more fully test distributed nodes is a significant problem.

HIL testing of DEC components is similar to HIL testing of general hardware components in that it is often preferable to perform bench or unit testing on individual components before doing a system-level, integration type of HIL test. Best practice dictates that testing should be an integral part of product development, and most of the tests conducted throughout development are unit-level tests. Once the individual components are satisfactory, system-level tests are performed, as they can determine if a fully integrated system works as intended. If such tests determine that the integrated system does not perform as intended, then the test results can often help identify and solve system-level integration issues. This work is concerned with a system-level test of a particular smart sensor integrated into a particular test loop. Unit tests of individual functions of this smart sensor have been performed at NASA before beginning this work, but the results of this previous unit testing are mainly proprietary and thus remain unpublished.

B. Objectives and Outline

There are two main objectives: 1) extend the wide bandwidth signal generating capability from a low order plant model, and 2) demonstrate a modular HIL test system that includes the ability to swap node function between hardware and simulation.

The development of a proof of concept HIL test is described in Section II, and the results of this test are given in Section III. Section IV provides discussion of an initial ancillary model of the high-bandwidth station 3 pressure (P_3) signal that is to be used for future work. Finally, conclusions and future work directions are given in Section V.

II. HIL Test Implementation

A. HIL Test Tools and Components

This paper explores concepts for testing smart, distributed engine components. The C-MAPSS40k engine system is used as the platform for the development of these concepts. C-MAPSS40k is a high fidelity model of a high bypass, 40,000 lbf thrust class turbofan engine written in MATLAB/Simulink, with some library components written in the C language.⁴ It is well suited for control systems research due to its transparency and ease of use, however, the architecture of C-MAPSS40k was not designed specifically with DEC system modeling in mind. To facilitate HIL testing of DEC systems, a real-time, distributed version of the C-MAPSS40k engine simulation was adapted from previous work.⁵

All of the C-MAPSS40k simulation runs conducted in this work use a simulation time step of 15 ms. For the purposes of the HIL tests described in this work, signal bandwidth distinctions are based on this time step value. Specifically, low bandwidth signals are defined as those that can be represented in a simulation running at 15 ms time steps. Due to the Nyquist sampling theorem, this means signals with a bandwidth, $f = 1 / (2 * 0.015) = 33.3$ Hz. Audible compressor stall precursors can be detected by the human ear.⁶ For the purposes of this work, high bandwidth signals are thus defined in this work to be signals with a bandwidth between 33.3 Hz and 20 kHz.

The HIL test platform computer system at NASA Glenn Research Center used for this work is called the distributed engine control system simulator (DECSS), which is a computer system intended for real-time distributed engine control research. The DECSS is a Concurrent Computer Corporation iHawk computer with 16 CPU cores, 32 GB of RAM, and peripherals such as a real-time clock, various analog and digital I/O, and transceivers for a variety of serial communication protocols including UDP over Ethernet, RS232, RS485, and CAN. It runs the Concurrent Red Hawk Linux operating system, which is based on CentOS/Red Hat Enterprise Linux with added real-time computing capabilities. The Concurrent development environment for real-time simulations is called Simulation Workbench (or Sim Workbench), and is the integrated development environment (IDE) that these HIL tests are developed in.

A smart pressure sensor that has wide-bandwidth capability is under development by Sporian Microsystems. This device has been made available to NASA for use in proof of concept HIL tests of DEC systems. This device is composed of a prototype smart node, as well as a proxy pressure transducer intended to interface the smart node to a HIL simulator. The smart node is a collection of electronics that digitizes and processes the pressure sensor data, and transfers that data to other smart nodes over a digital communication bus. The proxy pressure transducer replaces a hardware pressure transducer. Specifically, it takes an analog voltage input and interprets it as a simulated pressure signal. A proxy transducer is necessary, as the DECSS is a dry test bed. That is, there are no physical analog signals, such as fluid pressures, to measure. Electrical analogs of pressures are generated via an analog voltage output card instead. This allows for a much simpler and safer test environment at the expense of some of the test fidelity. This is acceptable in this case, as the main objectives are to demonstrate a proof-of-concept HIL test of a DEC smart node, and to test the processing capabilities and the algorithms in the smart node; they do not include testing the operation of any particular pressure transducer. The smart node and the proxy pressure transducer are combined and treated as a single black box device for the purposes of the HIL test in this work. This combined smart sensor device is considered to be the device under test (DUT). Note that “smart sensor” refers to this combined device throughout this paper whereas “smart node” refers to the electronics, independent of the proxy pressure transducer.

B. High-Level Formulation of HIL Test

In order to further describe this HIL test, a block diagram of the components in the test loop is shown in Figure 1. This diagram depicts the following: the DECSS computes and generates an analog output voltage representing P_3 data from the C-MAPSS40k engine system simulation. This signal is fed into the analog input of the proxy pressure transducer. This proxy transducer’s impedance varies with its analog input voltage just as a real transducer’s impedance would vary with pressure input, so in order to obtain pressure data, the smart node electronics block samples the proxy pressure transducer as it would a real pressure transducer. The smart node then computes fast Fourier transform (FFT) and time domain pressure data from the response of the proxy pressure transducer, and these

data are read back by the DECSS via a UDP-based communication interface. The data from the smart sensor are interpreted as a sensed P_3 signal, which is used for closed-loop control in the engine simulation. This means that the smart sensor is used in the simulation loop, and thus the DECSS and smart sensor together form a HIL system. Further, the fact that the smart sensor architecture uses a proxy pressure transducer is useful, because it means that a HIL test could be performed where the smart node instead connects to a real pressure transducer.

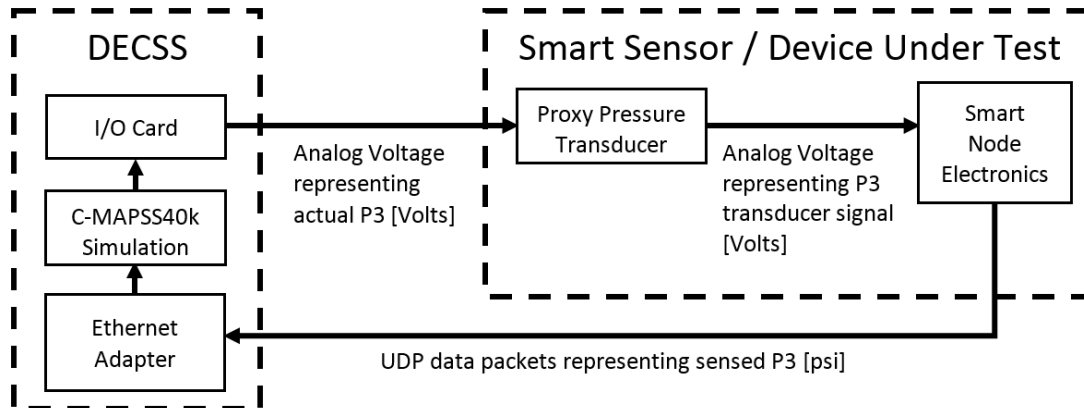


Figure 1. Block diagram of the HIL test system composed of the DECSS and DUT.

The smart node is capable of making multiple measurements, including wide bandwidth total pressure as well the traditional low bandwidth static pressure. This present HIL test work is limited to sensing station 3 static pressure, or P_{s3} . This is done to obtain preliminary results that prove the concept of the use of the DECSS for HIL tests by replacing a sensor model used by the original C-MAPSS40k controller with a hardware device. The C-MAPSS40k controller is designed to use P_{s3} in its W_f/P_{s3} (i.e., fuel flow rate/ P_{s3}) min limiter, which protects low-pressure compressor surge margin and fuel-to-air ratio design limits. It also features P_{s3} min and P_{s3} max limit logic. The value of this work is that, by performing the steps necessary to replace a simulated P_{s3} sensor with smart sensor hardware and running a HIL test, one can extrapolate a general procedure for HIL testing of DEC smart nodes.

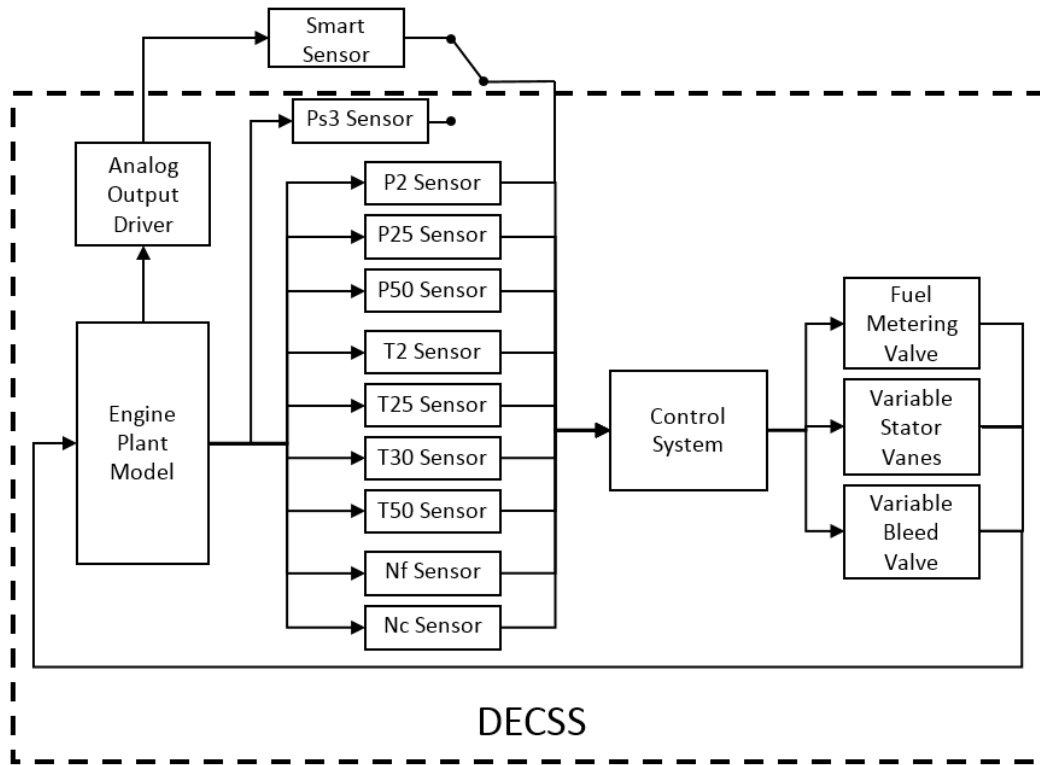


Figure 2. Block diagram of the distributed C-MAPSS40k simulation as it runs on the DECSS. A switch is used to control whether the simulated *Ps3* signal or the *Ps3* signal from the smart sensor is used for controller feedback. The items within the box drawn with a thick dashed line are all simulation or hardware components contained within the DECSS.

A block diagram showing how the smart sensor integrates into the distributed C-MAPSS40k system running on the DECSS is shown in Figure 2. Note that the switch shown in this figure is implemented as an *if* statement that checks for a HIL test enable flag in the smart node UDP communication C code. If the flag is set, then the simulated *Ps3* sensor data is blocked and overwritten with the data read back from the smart sensor via UDP; otherwise, the simulated *Ps3* sensor data is passed through and used for feedback.

C. Design of HIL Test using Sim Workbench

Sim Workbench provides useful, drag-and-drop style graphical tools for taking Simulink-based simulation systems, building them for the Concurrent Red Hawk platforms (in this case, the DECSS), and creating and running real-time test schedules. In order to create a test scenario in Sim Workbench, a real-time database (RTDB) must first be created. An RTDB is a data structure that is used to share data between the different programs running concurrently in a given test.

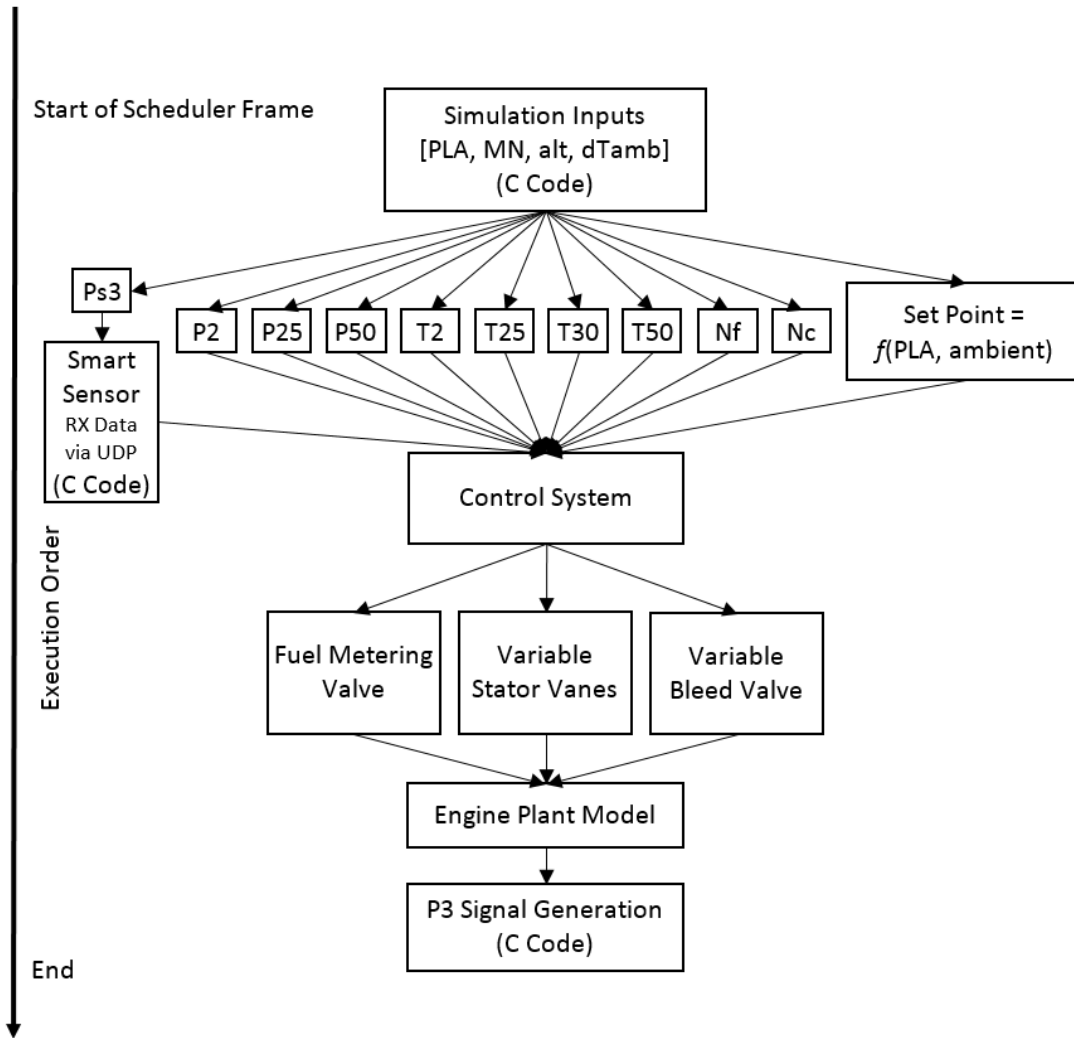


Figure 3. Block diagram of the Sim Workbench test used in this work, showing the programs that constitute this test (in boxes). This diagram is drawn in a similar fashion to the “tests” screen in Sim Workbench. Programs written in the C language are indicated with the text, “(C Code).” All other programs are generated from Simulink model reference blocks. Note that the arrows in this figure indicate data dependency in the simulation loop. This means that a program that points to another program will be scheduled to execute first, before the program it points to, because the second program (the one being pointed to) needs the results of the first program in order to run.

After creating the RTDB, the programs that are part of the test must be included into the Sim Workbench project and built. There are a variety of programming languages that can be used to create Sim Workbench tests, but in the case of this HIL test, these programs are the distributed C-MAPSS40k Simulink block diagrams (individual sensors and actuators, the engine plant, and control system), as well as some C programs used to interface the C-MAPSS40k simulation with the smart sensor hardware, generate input profiles, and save simulation output variables. Simulink programs are developed on a Windows machine and pushed over the local area network (LAN) to the DECSS, where they are built. During this build process, RTDB variables are created based on Simulink block naming conventions; these variables are the only link between individual programs in a test.

After the build process, a Sim Workbench “test” must be created; a “test” is a schedule that describes the execution order of programs in each simulation time step. In the Sim Workbench IDE, this is accomplished by adding blocks that correspond to individual programs, and drawing arrows connecting them that indicate data dependency relationships between them (e.g., the control system depends on the RTDB variables containing output data from each of the sensors). Once created, the test can be run. Sim Workbench also includes tools for creating graphical user interfaces (GUIs) that help users run tests and observe test variables, although these GUIs are not strictly necessary to

run tests. A block diagram showing the construction of the Sim Workbench test used for this work is included in Figure 3.

This diagram does not show the frequency divider, frequency multiplier, or other multi-rate controls that are available in the test creation screen of the SimWorkbench IDE, as these were not used to create the test used in this work. Note that the programs corresponding to the blocks in Figure 3 each execute once per scheduler frame (i.e., simulation time step).

The first block to execute each frame is the “Simulation Inputs” block, which drives the RTDB variables containing simulation inputs. Next to execute are the sensor blocks and the block that generates controller setpoint values from the power lever angle (PLA) profile (for either the engine pressure ratio or fan speed (EPR or N_f) controller). The sensors and setpoint blocks all execute in parallel on separate processor cores. Note that the smart sensor UDP block represents the C code that reads pressure data from the smart sensor via its UDP command application program interface (API). If the HIL test enable flag mentioned regarding the switch in Figure 2 is set, the smart sensor UDP block will overwrite the $Ps3$ simulation data with the data obtained from the smart sensor. If the HIL enable flag is set to false, the $Ps3$ simulation data will pass through and will be used for closed-loop control. After the sensors execute, the control system executes, and then the actuators, and then the engine plant model. Finally, the analog output signal representing $Ps3$ is generated. This drives the smart sensor’s pressure signal for the next time step. The next section describes how this test was conducted and shows the results obtained from it.

III. Preliminary HIL Test Results

For this HIL simulation, a burst and chop power lever angle (PLA) profile is provided to the engine at sea-level-static (SLS), which corresponds to an altitude of 0 ft, a Mach number of 0, and a standard-day temperature of 59 °F. Figure 4 compares data from a baseline simulation (the distributed C-MAPSS40k with a software model of a $Ps3$ sensor) indicated with blue lines, and data from the HIL test with the hardware smart sensor shown with red dashed lines. Specifically, the data shown in the plot are net thrust, fuel flow rate, actual $Ps3$ (i.e., the truth value of $Ps3$ produced by the C-MAPSS40k engine plant) and sensed $Ps3$ (i.e., either the output of C-MAPSS40k’s $Ps3$ sensor component, shown in blue, or the data read back from the smart sensor via UDP, shown as a red dashed line).

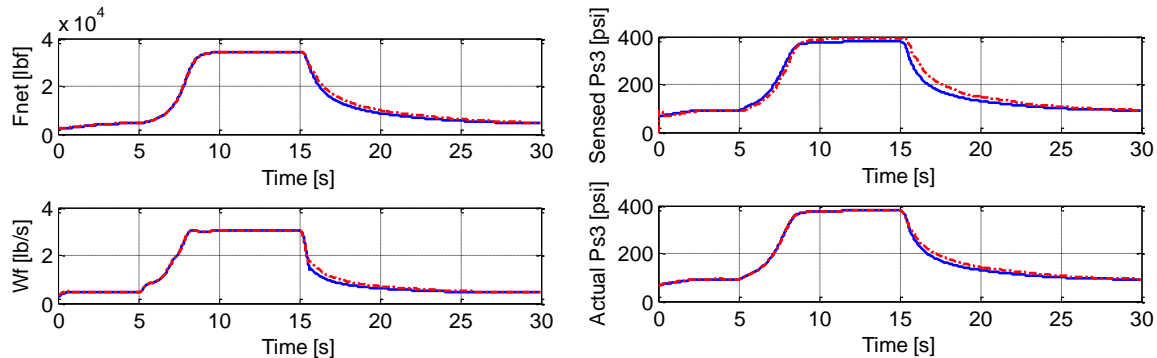


Figure 4. Simulation results showing net thrust, fuel flow rate, sensed $Ps3$, and actual $Ps3$, for distributed C-MAPSS40k with a simulated $Ps3$ sensor (solid blue line), and with the smart sensor hardware in the control loop (dash-dotted red line).

The data from the HIL case matches the simulation case well, especially during the trimming (0 to 5 s), throttle burst (5 to 10 s), and high throttle steady-state (10 to 15 s) periods of simulation. The only noticeable discrepancies before 15 s are in the sensed $Ps3$ plot. These are expected, because the smart sensor’s performance is not exactly the same as that of the $Ps3$ sensor contained in the C-MAPSS40k simulation.

Small differences in the net thrust (F_n), W_f , and actual $Ps3$ data can be seen in the plots in Figure 4 during decel (15 s to end of simulation)—before decel, the data match perfectly for these variables. This is because sensed $Ps3$ only begins to affect the closed loop response of the system when one of the $Ps3$ -based safety limiters becomes active, in this case, the $W_f/Ps3$ min limiter. The cause of the differences is that there is more $Ps3$ sensor signal lag in the HIL test case than there is in the baseline (simulated sensor) case. This may be caused by one of several things. First, the UDP communication channel between the smart sensor and the DECSS may be significantly slower than the communication channel that will be used in a practical implementation. Another is that the lag may be due to dynamics in the smart node, rather than delay in the communication channel. In general, the effects of the increased $Ps3$ signal lag in the HIL test case are small. A pilot would not likely notice a difference in the thrust response, and, more

importantly, the safety margins are all protected successfully in both cases. These results demonstrate that the smart node can be used successfully in a C-MAPSS40k style engine.

IV. $P3_{HB}$ Signal Model Development

This section discusses preliminary modeling and simulation results in MATLAB/Simulink of a closed-loop active surge control scheme that estimates HPC surge margin from high bandwidth $P3$ data. This work is discussed in anticipation of upcoming high bandwidth HIL tests of the smart node. As mentioned previously, the update rate of C-MAPSS40k is too slow to include the higher frequency dynamics characteristic of surge precursors. Therefore, an ancillary high bandwidth $P3$ model is added to C-MAPSS40k that incorporates high pressure compressor (HPC) surge margin information into the $P3$ signal in the form of pressure disturbances created in the HPC as the blades of the HPC rotor pass by the HPC stator vanes. These are referred to as blade passing pressure disturbances (BPPDs) throughout this paper. To model BPPDs, it is first assumed that a single smart $P3$ sensor is present on the engine and that it detects BPPDs from only one stage of the HPC. It is then assumed that these pressure disturbances are sinusoidal and all of the noise present in the signal can be lumped together and assumed to be zero-mean additive white Gaussian noise, $N(0, \sigma^2)$. Further, it is assumed that the frequency of the BPPD signal is proportional to the speed of the high pressure or core shaft, Nc . Further, the magnitude of the sinusoidal BPPD signal is assumed to be related to the HPC surge margin values from the underlying C-MAPSS40k model via a smooth nonlinear function. These assumptions are made, considering both first principles and qualitative observations of AC coupled compressor data (i.e., with the average signal value or DC component removed).⁶

One possible model for a high bandwidth $P3$ signal that satisfies these assumptions is

$$P3_{HB} = P3_S + \left(\frac{k_2 [1 - \tanh(k_3 * (SM_{HPC} - k_4))]}{2} \cdot \cos(k_1 \cdot 2\pi Nc \cdot t) \right) + N(0, \sigma^2) \quad (1)$$

where $P3_{HB}$ is the ancillary high bandwidth $P3$ signal model, SM_{HPC} and Nc are the HPC surge margin and core speed variables, respectively, from C-MAPSS40k, and k_1, k_2, k_3, k_4 , and σ are all model parameters. k_1 is the proportionality constant relating Nc to the BPPD signal frequency, k_2 is the maximum magnitude that the BPPD signal will obtain as HPC surge margin is varied, k_3 controls how quickly the BPPD magnitude rolls off as HPC surge margin is decreased, and k_4 controls the HPC surge margin value at which the BPPD signal will have half its maximum magnitude. Further, $\tanh(\cdot)$ represents the hyperbolic tangent function. The $\tanh(\cdot)$ function is chosen as the nonlinearity that relates HPC surge margin to the BPPD magnitude because the true relationship is unknown, although it is assumed to have a smooth sigmoidal shape—the $\tanh(\cdot)$ function is chosen as it is representative of the general class of sigmoidal functions.

Problems with the assumptions above include the fact that a real HPC may have many stages with different numbers of blades, and this means that there may be many harmonics present in the pressure signal measured at any given location in the HPC. Further, surge and pre-surge are phenomena with spacial dynamics, and so a zero-dimensional model like this one (i.e, C-MAPSS40k with a high bandwidth ancillary model driven by low bandwidth simulation data) can only approximate them.

A simplistic approach to implementing this model for purposes of simulating the smart $P3$ sensor purely in software is to generate the high-bandwidth pressure signal in the time domain and use it as feedback where necessary in the simulation. This is not desirable, because the simulation step size would need to be lowered below the Nyquist sampling period given the highest signal frequency to be represented in the simulation, and so this would make the simulation run prohibitively slowly. To avoid this, the following approach is used: at every simulation time step, a short window of the time domain high bandwidth pressure signal is generated, and the FFT of the signal is computed over this window. This window comprises M samples, using a sampling period of T_s . Then, an estimate of the BPPD signal magnitude can be computed by sampling the FFT data at frequency $k_1 \cdot Nc$.

The FFT-based approach for modeling the BPPD magnitude sensor is chosen over simpler approaches (such as a single nonlinear function that takes in simulation variables and directly outputs the BPPD magnitude), because it more closely reflects operation of the hardware HIL system using the smart pressure sensor. FFT data is used in the algorithms running in the physical smart sensor to estimate BPPD magnitude, and so the FFT step should not be omitted from the BPPD sensor simulation model as doing so constitutes neglecting the errors introduced by the quantization inherent in FFT binning.

In order to use these BPPD signal magnitudes for closed-loop control, an observer model must be made that predicts the underlying system state (HPC surge margin) using sensed data (BPPD magnitude). For preliminary control studies, it is proposed that a simple polynomial curve fit model will suffice. Such a model is made from empirical simulation data obtained by running C-MAPSS40k with the ancillary $P3_{HB}$ and smart sensor models, and fitting a

curve that relates the resulting FFT sample magnitude (estimated BPPD magnitude) to HPC surge margin. This formula obtained via curve fitting is used to obtain HPC surge margin estimates for feedback in the simulation.

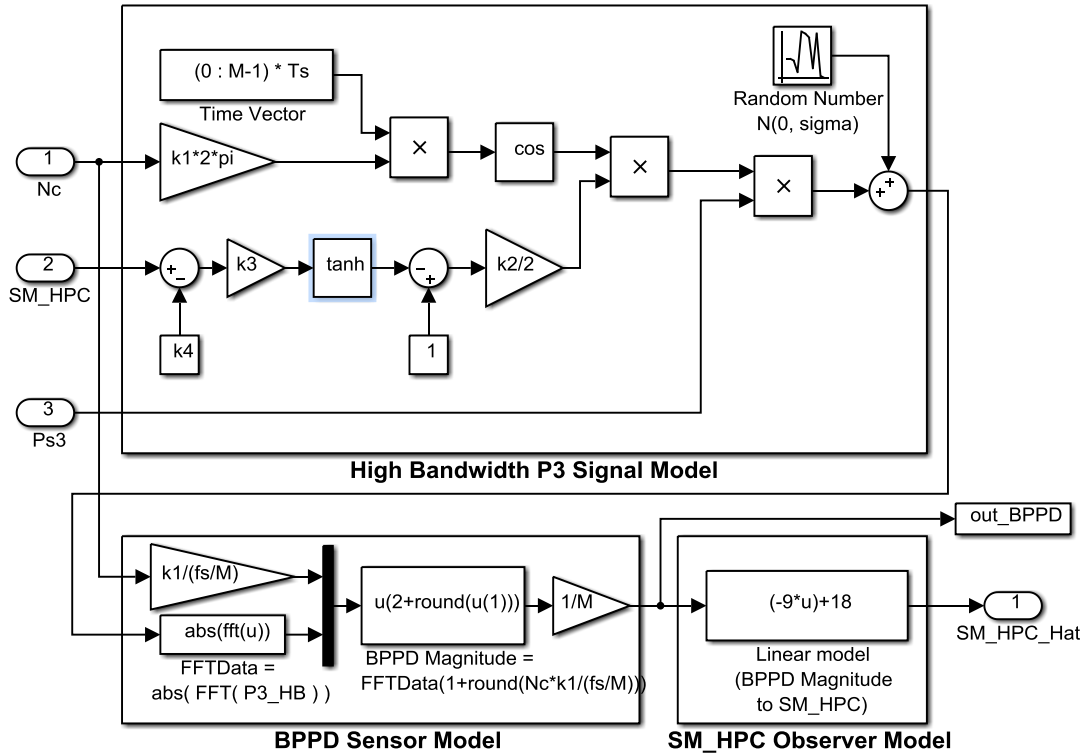


Figure 5. Block diagram of the ancillary modeling components added to C-MAPSS40k to enable BPPD-based active HPC surge control, showing the inputs from C-MAPSS40k in the top left, and the estimated HPC surge margin output in the bottom right. The variable, u , in the blocks in this diagram refers to the input signal of that particular block.

Additionally, a maximum W_f limit regulator designed to protect HPC surge margin during throttle transients is constructed in a similar fashion to the other limit regulators that exist in C-MAPSS40k.⁷ One such limiter that generates a maximum fuel flow limit by comparing the current HPC surge margin estimate (computed from BPPD magnitude) with a desired surge margin limit value has been created and added to C-MAPSS40k as part of this modeling development. Figure 5 shows a block diagram of the modeling components added to C-MAPSS40k, as described above. The limit regulator logic is not shown, as this logic is analogous to that of other limit regulators in baseline C-MAPSS40k.⁷ Note that fs and T_s , (where $fs = 1 / T_s$) in this diagram are the sampling frequency and sampling period of the high bandwidth system.

Plots of representative simulation results for the $P3_{HB}$ -based active surge control simulation are shown in Figure 6. Note that these results were obtained completely in simulation without the use of sensor hardware. A HIL test using physical smart node hardware is reserved for future work. The simulation condition for this test is a burst-and-chop PLA profile at SLS. The model parameters for this simulation were chosen as follows: number of blades, $n_B = 20$, $k_1 = n_B/60$ [blades/(seconds/minute)], $k_2 = 1$ [psi], $k_3 = 0.2$ [-], $k_4 = 12$ [% HPC surge margin], $T_s = 100$ [μ s], $f_s = 10$ [kHz], $M = 8192$ [time steps or bins]. The HPC surge margin limit used in the limit regulator was chosen to be 11%. These are arbitrary choices, picked to evaluate the closed-loop system in a general manner. The number of compressor blades was chosen to be a small value of 20 so that the maximum BPPD signal frequency $f_{MAX} = N_{C_{MAX}} * n_B$, is less than 4 kHz, which is the maximum signal frequency that can be resolved by the physical smart sensor being modeled in this work.

It was found that a linear model, $y(u) = -9u + 18$, approximates the relationship between BPPD magnitude (y) and HPC surge margin (u) well when the engine is operating at SLS over transients that produce HPC surge margins in the neighborhood of 12%.

Note that the HPC surge margin signal in Figure 6 shows oscillatory components. This is because the HPC surge margin limiter is chattering between active and inactive states. This behavior can be improved by tuning the HPC surge margin limiter—the limiter parameters were simply chosen to match those from the C-MAPSS40k N_c acceleration limiter. Also note that the model does not predict surge margins above 20%, however, this is not the goal. It is most important that the model predict HPC surge margin accurately at values significantly below 20%, where the HPC approaches surge during throttle burst transients, so that the fuel flow transient can be limited or slowed down such that a given HPC surge margin can be protected. Note that the HPC surge margin produced by the observer logic is currently designed at SLS. It can easily be extended to cover the entire C-MAPSS40k flight envelope by scheduling the observer model parameters as functions of ambient conditions.

Future work involves finding high quality empirical data that relates compressor pre-surge phenomena to the high frequency components of $P3$, and using these data to produce and validate a higher fidelity model. Much of the compressor pressure data showing stall precursors found in the literature is presented as relative, or AC-coupled pressure data, without information about DC pressure levels, and often without specified units.⁶ More complete data would allow more realistic models of pre-surge phenomena to be constructed. Other work to be done on the active surge control application of the $P3_{HB}$ model includes improvement of the HPC surge margin observer and limit regulator models to enhance the efficacy of the controller and ensure that HPC surge margins are protected consistently over the entire flight envelope.

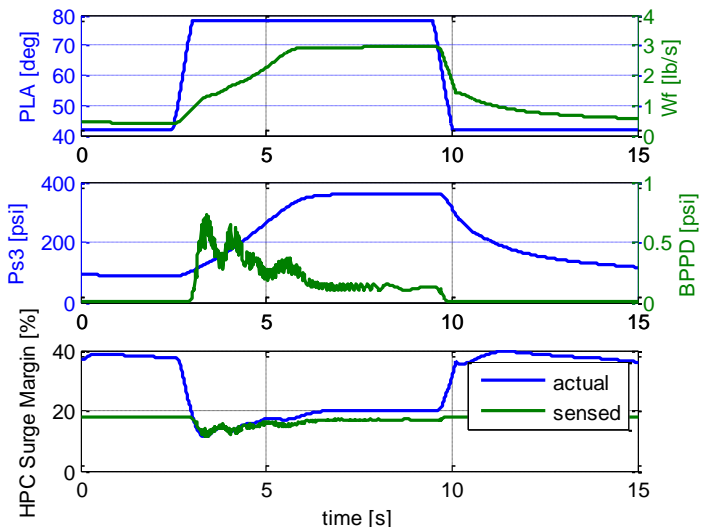


Figure 6. Plots showing PLA profile and resulting fuel flow, $Ps3$ and magnitude of the BPPD signal, and actual and sensed HPC surge margin. $Ps3$ is shown because it is a sensed engine variable that helps show the scale of the BPPD signal.

V. Conclusions

A low bandwidth hardware-in-the-loop (HIL) test of a hardware station 3 pressure ($P3$) smart sensor was conducted on the distributed engine control system simulator (DECSS) real-time computing platform with the C-MAPSS40k engine model. The development of this HIL test demonstrates a process for replacing an individual software component in a large scale system simulation with its functional hardware equivalent. The results of the HIL simulation show that a hardware $P3$ smart sensor can be used successfully as the $Ps3$ sensor for closed loop control in a C-MAPSS40k engine system. Further, a preliminary ancillary model for generating a high bandwidth $P3$ signal has been described for use in future work. In this model, it is assumed that the pressure signal components due to high-pressure compressor (HPC) blades passing by a sensor location are a nonlinear function of the static pressure at that

location (P_{s3}), the HPC surge margin, and the speed of the HPC. This model was used in a software-only simulation test where C-MAPSS40k was augmented with this high-bandwidth $P3$ signal model, a blade passing pressure disturbance (BPPD) smart sensor model, a HPC surge margin observer model, and a HPC surge margin limit regulator. Results from this test demonstrate that closed-loop control of HPC surge margin can be achieved with these components, and these results provide a baseline with which to compare future HIL tests of the same active surge control concepts. Future work directions are discussed—these include development of more realistic surge modeling and control techniques, as well as other applications of these HIL testing tools.

Acknowledgments

Funding and resources for this work are provided by NASA Glenn Research Center and the Transformative Aeronautics Concepts Program, Transformational Tools and Technologies Project. Also, the authors would like to thank Sporian Microsystems for providing hardware, documentation, and support outside the scope of their contracted work to enable the HIL test published in this paper.

References

- ¹Culley, D., “Transition in Gas Turbine Control System Architecture: Modular, Distributed, and Embedded,” *ASME Turbo Expo 2010: Power for Land, Sea, and Air, Vol. 3*, Glasgow, Scotland, United Kingdom, June 2010, pp. 287–297.
- ²Bright, M. M., Culley, D. E., Braunscheidel, E. P., and Welch, G. E., “Closed Loop Active Flow Separation Detection and Control in a Multistage Compressor,” *43rd AIAA Aerospace Sciences Meeting and Exhibit*, Reno, Nevada, 2005, pp. 1039-1050.
- ³Wadia, A. R., Christensen, D., and Prasad, J. V. R., “Compressor Stability Management in Aircraft Engines,” *Proceedings of the 25th Congress of the International Council of the Aeronautical Sciences*, Hamburg, Germany, 2006.
- ⁴May, R.D., Csank, J., Lavelle, T.M., Litt, J.S., and T.H., Guo, “A High-Fidelity Simulation of a Generic Commercial Aircraft Engine and Controller,” NASA/TM-2010-216810, October 2010.
- ⁵Zinnecker, A. M., Culley, D. E., and Aretskin-Hariton, E. D., “A Modular Framework for Modeling Hardware Elements in Distributed Engine Control Systems,” *Proceedings of the 50th AIAA Joint Propulsion Conference*, Cleveland, Ohio, USA, 2014.
- ⁶Abdel-Fattah, A. M. and Vivian, A. S., “Development of the Larzac Engine Rig for Compressor Stall Testing,” Defense Science and Technology Organization, Victoria, Australia, DSTO-RR-0377, 2010.
- ⁷Csank, J., May, R.D., Litt, J.S., and Guo, T.H., “Control Design for a Generic Commercial Aircraft Engine,” *2010 AIAA Joint Propulsion Conference*, Nashville, Tennessee, 2010.